ДОМАШНА ЗАДАЧА БРОЈ 1 И 2

-СОФТВЕРСКИ КВАЛИТЕТ И ТЕСТИРАЊЕ-

Изработил:

Костадин Љаткоски (161005)

Професор:

д-р Бојана Котеска

1. Цел на домашната задача

Целта на оваа домашна задача е да се креира метод кој проверува дали еден број е степен на бројот 2. Методот враќа променлива Boolean која има вредност true доколку бројот (влезен параметар на методот) е степен на бројот 2, а во спротивно враќа false доколку бројот (влезен параметар на методот) не е степен на бројот 2. Влезниот параметар е од тип Integer.

Потоа, користејќи JUnit е потребно:

- Да се креираат тестови за дадениот метод;
- Да се одговори дали со креираните тестови се откриваат можните недостатоци и испади на напишаниот метод;
- Да се поправат тестовите за да се откријат недостатоците и испадите на напишаната функција;
- Да се креираат параметризирани тестови;
- Да се креираат тестови за исклучоци.

2. Имплементација на методот кој проверува дали еден број е степен на бројот 2

```java
public class PowerOfTwoChecker {
    public Boolean isPowerOfTwo(Integer num) {
        if (num == null)
            throw new IllegalArgumentException("Argument must not be null");
        if(num % 2 != 0)
            return false;
        while (num % 2 == 0)
            num /= 2;
        return num == 1;
    }
}
```

Идејата на имплементацијата е следнава:

- Доколку параметарот има вредност ***null***, фрли исклучок од тип ***IllegalArgumentException*** со дадена порака;
- Доколку бројот е непарен, веднаш врати ***false***;

- Сè додека бројот е делив со 2, дели го бројот. Доколку бројот е степен на 2, на крај ќе добиеме резултат 1. Провери дали бројот по неколкукратното делење со 2 дали има вредност 1. Доколку вредноста е 1, врати ***true***, а во спротивно врати ***false***.

3. Генерирање на тестови за дадениот метод (класа PowerOfTwoCheckerTests)

Креирав класа ***PowerOfTwoCheckerTests*** во која ќе ги пишувам тестовите за напишаниот метод.

```
1   import org.junit.jupiter.api.BeforeEach;
2   import org.junit.jupiter.api.Test;
3
4   import static org.junit.jupiter.api.Assertions.*;
5
6   public class PowerOfTwoCheckerTests {
7
8       private PowerOfTwoChecker powerOfTwoChecker;
9
10      @BeforeEach
11      public void setup() {
12          powerOfTwoChecker = new PowerOfTwoChecker();
13      }
14
```

Методот ***public void setup()*** е анотиран со анотацијата ***@BeforeEach*** што значи дека овој метод ќе се извршува пред секој тест. Методот е многу едноставен- пред секој тест инстанцира објект од класата ***PowerOfTwoChecker*** во која го напишавме методот ***public Boolean isPowerOfTwo(Integer num).***

За првите неколку тестови кои следат, имам напишано по 2 тест методи кои во суштина се идентични, но само користат различни ***assert*** методи за демонстрациски цели, како и за вежбање. Имено користам и ***assertTrue/assertFalse*** и ***assertEquals/assertNotEquals***.

3.1. Со овој тест сакаме да го видиме однесувањето на методот isPowerOfTwo и вредноста која очекуваме да ја добиеме како резултат е ***true*** за број кој е степен на бројот 2, па затоа како влезна вредност е избрана вредноста **32**.

```
15      @Test
16      public void testIsPowerOfTwo1() {
17          assertTrue(powerOfTwoChecker.isPowerOfTwo( num: 32));
18      }
19
20      @Test
21      public void testIsPowerOfTwo1_1() {
22          assertEquals( expected: true, powerOfTwoChecker.isPowerOfTwo( num: 32));
23      }
```

3.2. Со овој тест сакаме да го видиме однесувањето на методот isPowerOfTwo и вредноста која очекуваме да ја добиеме како резултат е ***false*** за непарен број, па затоа како влезна вредност е избрана вредноста **55**.

```
25      @Test
26      public void testIsPowerOfTwo2() {
27          assertFalse(powerOfTwoChecker.isPowerOfTwo( num: 55));
28      }
29
30      @Test
31      public void testIsPowerOfTwo2_2() {
32          assertNotEquals( unexpected: true, powerOfTwoChecker.isPowerOfTwo( num: 55));
33      }
```

3.3. Со овој тест сакаме да го видиме однесувањето на методот isPowerOfTwo и вредноста која очекуваме да ја добиеме како резултат е ***false*** за позитивен парен број кој не е степен на бројот 2, но неколкукратно е делив со 2, па затоа како влезна вредност е избрана вредноста **200**.

```
35      @Test
36      public void testIsPowerOfTwo3() {
37          assertFalse(powerOfTwoChecker.isPowerOfTwo( num: 200));
38      }
39
40      @Test
41      public void testIsPowerOfTwo3_2() {
42          assertEquals( expected: false, powerOfTwoChecker.isPowerOfTwo( num: 200));
43      }
```

3.4. Со овој тест сакаме да го видиме однесувањето на методот isPowerOfTwo и вредноста која очекуваме да ја добиеме како резултат е ***false*** за негативен парен број кој не е степен на бројот 2, но неговата апсолутна вредност е степен на бројот 2, па затоа како влезна вредност е избрана вредноста **-512**.

```
45      @Test
46      public void testIsPowerOfTwo4() {
47          assertFalse(powerOfTwoChecker.isPowerOfTwo( num: -512));
48      }
49
50      @Test
51      public void testIsPowerOfTwo4_2() {
52          assertNotEquals( unexpected: true, powerOfTwoChecker.isPowerOfTwo( num: -512));
53      }
```

3.5. Со овој тест сакаме да го видиме однесувањето на методот isPowerOfTwo и вредноста која очекуваме да ја добиеме како резултат е ***false*** и во овој случај имаме гранична вредност, односно бројот **0**.

```
56      @Test
57      public void testIsPowerOfTwo5() {
58          assertFalse(powerOfTwoChecker.isPowerOfTwo( num: 0));
59      }
60
61      @Test
62      public void testIsPowerOfTwo5_2() {
63          assertNotEquals( unexpected: true, powerOfTwoChecker.isPowerOfTwo( num: 0));
64      }
```

Кога го стартував тестот, увидов дека тестот скоро една минута не завршува, па сфатив дека за влезна вредност **0**, методот кој го напишав никогаш не излегува од *while* циклусот. Односно **софтверскиот недостаток (software failure)** во случајов е тоа што не правам никаква проверка пред влезот во *while* циклусот, па условот ***while (num % 2 == 0)*** е секогаш исполнет, а во следниот чекор ***num /= 2*** секогаш дава резултат 0. Според тоа токму тука се пропагира софтверскиот недостаток и резултира со **софтверска грешка (software error)**. Во случајов тестот се покажува како добар бидејќи се случи **софтверски испад (software failure)**, ние очекувавме да добиеме резултат *false*, но извршувањето на нашиот метод не завршува.

3.6. Со овој тест сакаме да го видиме однесувањето на методот isPowerOfTwo и вредноста која очекуваме да ја добиеме како резултат е ***true*** и во овој случај имаме гранична вредност, односно бројот **1** кој е единствениот непарен број кој е степен на бројот 2.

```
66      @Test
67      public void testIsPowerOfTwo6() {
68          assertTrue(powerOfTwoChecker.isPowerOfTwo( num: 1));
69      }
70
71      @Test
72      public void testIsPowerOfTwo6_2() {
73          assertEquals( expected: false, powerOfTwoChecker.isPowerOfTwo( num: 1));
74      }
```

Кога го извршив тестот, добив различен резултат од она што го очекував. Односно **софтверскиот недостаток (software failure)** во случајов е тоа што проверувам дали бројот е непарен со наредбата ***if (num %2 != 0)*** и доколку условот е исполнет, во следната линија веднаш враќам ***false***, односно дека бројот не е број кој е степен на 2. Според тоа токму тука се пропагира софтверскиот недостаток и резултира со **софтверска грешка (software error)**. Во случајов тестот се покажува како добар бидејќи се случи **софтверски испад (software failure)**, ние очекувавме да добиеме резултат *true*, но добивме *false*.

4. Генерирање на тестови за даденият метод (класа PowerOfTwoCheckerTimeoutTests)

```
9   public class PowerOfTwoCheckerTimeoutTests {
10
11      private PowerOfTwoChecker powerOfTwoChecker;
12      private Long startTime;
13
14      @BeforeEach
15      public void setup() {
16          powerOfTwoChecker = new PowerOfTwoChecker();
17          startTime = System.nanoTime();
18      }
19
20      @AfterEach
21      public void methodExecutionTime() {
22          long timeElapsed = System.nanoTime() - startTime;
23          System.out.println("Approximately Execution time in milliseconds: " + timeElapsed / 1000000);
24      }
```

Методот ***public void setup()*** е анотиран со анотацијата ***@BeforeEach*** што значи дека овој метод ќе се извршува пред секој тест. Методот е многу едноставен- пред секој тест инстанцира објект од класата ***PowerOfTwoChecker*** во која го напишавме методот ***public Boolean isPowerOfTwo(Integer num).*** Исто така имаме и една приватна променлива ***Long startTime*** која ја користиме за да го земеме моменталното време пред извршувањето на тестот.

Методот ***public void methodExecutionTime()*** е анотиран со анотацијата ***@AfterEach*** што значи дека овој метод ќе се извршува после секој тест. Во овој метод во локалната променлива ***long timeElapsed*** всушност сместуваме апроксимација на времето потребно тестот да се изврши (од моменталното време, го одземаме времето сместено во променливата ***startTime*** која ја иницијализиравме во методот ***setup***. Потоа го печатиме времето потребно за тестот да заврши во милисекунди.

Ова го правевме сè со цел да имаме некаква претстава колку време е потребно за да се изврши методот кој го напишавме, бидејќи со следниве два теста токму тоа ќе го провериме, користејќи ***assertTimeout***.

Со овие два теста сакаме да го видиме однесувањето на методот isPowerOfTwo, односно дали извршувањето на методот е подолго од ***15 милисекунди***. Во првиот тест како влезна вредност го имаме бројот **2**, што доколку се извршува делот од кодот каде се врши кратење на бројот со 2, односно *while* циклусот, бројот на итерации би бил најмал. Во вториот тест пак како влезна вредност го имаме бројот **1073741824** што пак е најголемиот 32 битен Integer кој е степен на бројот 2, па тогаш во *while* циклусот би имале најголем број на итерации.

```
26      @Test
27      public void testIsPowerOfTwoTimeout1() {
28          assertTimeout(Duration.ofMillis(15), () -> powerOfTwoChecker.isPowerOfTwo( num: 2));
29      }
30
31      @Test
32      public void testIsPowerOfTwoTimeout2() {
33          assertTimeout(Duration.ofMillis(15), () -> powerOfTwoChecker.isPowerOfTwo( num: 1073741824));
34      }
```

5. Генерирање на тестови за дадениот метод (класа PowerOfTwoCheckerParameterizedTest)

```java
public class PowerOfTwoCheckerParameterizedTest {

    private PowerOfTwoChecker powerOfTwoChecker;

    @BeforeEach
    public void setup() {
        powerOfTwoChecker = new PowerOfTwoChecker();
    }

    public static Collection<Object[]> inputValues() {
        return Arrays.asList(new Object[][] {
                {32, true},
                {55, false},
                {200, false},
                {-512, false},
                {0, false},
                {1, true}
        });
    }

    @ParameterizedTest
    @MethodSource("inputValues")
    public void isPowerOfTwoParameterizedTest(int inputNum, boolean expectedResult) {
        assertEquals(expectedResult, powerOfTwoChecker.isPowerOfTwo(inputNum));
    }
}
```

Методот ***public void setup()*** е анотиран со анотацијата ***@BeforeEach*** што значи дека овој метод ќе се извршува пред секој тест. Методот е многу едноставен- пред секој тест инстанцира објект од класата ***PowerOfTwoChecker*** во која го напишавме методот ***public Boolean isPowerOfTwo(Integer num).***

Користиме параметризирани тестови сè со цел да не правиме посебен тест за секоја вредност на влезниот параметар, туку тие вредности можат да бидат сместени во некаква колекција, фајл итн.., па за секоја од нив ќе биде извршен соодветниот тест. Тука ги имаме истите вредности за влезниот параметар како и во првите 6 теста од оваа домашна задача.

6. Генерирање на тестови за даденият метод (класа PowerOfTwoCheckerExceptionTests)

```java
public class PowerOfTwoCheckerExceptionTests {

    private PowerOfTwoChecker powerOfTwoChecker;

    @BeforeEach
    public void setup() {
        powerOfTwoChecker = new PowerOfTwoChecker();
    }

    @Test
    public void isPowerOfTwoShouldThrowAnExceptionTest() {
        assertThrows(IllegalArgumentException.class, () -> powerOfTwoChecker.isPowerOfTwo( num: null));
    }

    @Test
    public void isPowerOfTwoShouldThrowAnExceptionTestWithMessage() {
        String exceptionMessage = "Argument must not be null";
        assertThrows(IllegalArgumentException.class, () -> powerOfTwoChecker.isPowerOfTwo( num: null), exceptionMessage);
    }

    @Test
    public void isPowerOfTwoShouldNotThrowAnExceptionTest() {
        assertDoesNotThrow(() -> powerOfTwoChecker.isPowerOfTwo( num: 64));
    }
}
```

Методот ***public void setup()*** е анотиран со анотацијата ***@BeforeEach*** што значи дека овој метод ќе се извршува пред секој тест. Методот е многу едноставен- пред секој тест инстанцира објект од класата ***PowerOfTwoChecker*** во која го напишавме методот ***public Boolean isPowerOfTwo(Integer num).***

Идејата на овие тестови е да го видиме однесувањето на методот доколку имаме **null** вредност како влезен параметар. Бидејќи во методот кој го напишавме, правевме проверка дали влезниот параметар е *null*, и доколку е, фрлавме исклучок – ***IllegalArgumentException***. Друга опција беше исклучокот *NullPointerException*, но ова семантички некако повеќе одговара во ситуацијава. Во вториот тест дури проверуваме дали исклучокот има соодветна порака (***exceptionMessage***) која ја задаваме во конструкторот на исклучокот. Во последниот тест очекуваме дека нема да биде фрлен никаков исклучок.

7. Корекција на методот врз основа на откриените недостатоци и испади со напишаните тестови

Со досега напишаните тестови, увидовме испади на програмата кога како влезен параметар ги имавме вредностите **0** и **1**.

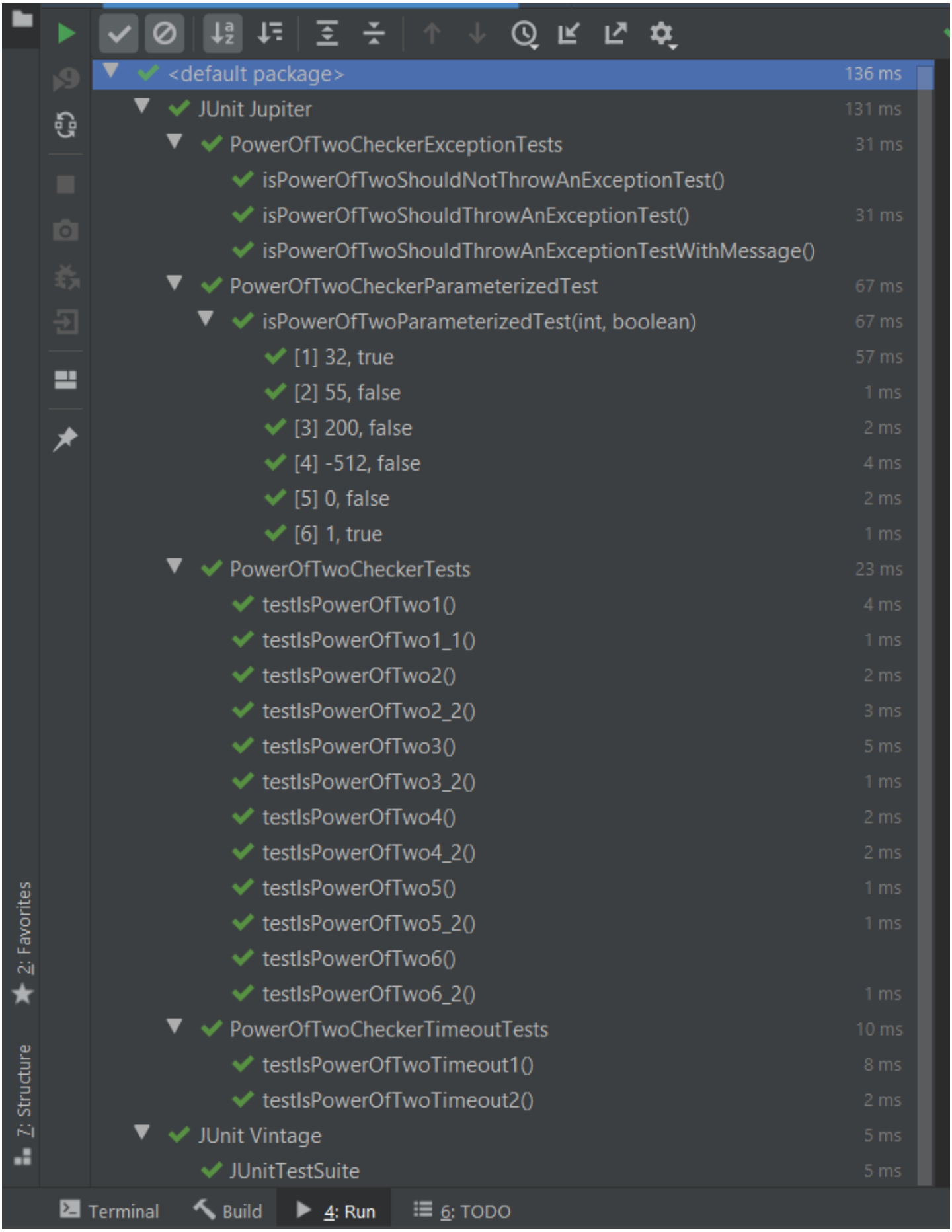
Корегирана имплементација на метод:

```
public class PowerOfTwoChecker {
    public Boolean isPowerOfTwo(Integer num) {
        if(num == null)
            throw new IllegalArgumentException("Argument must not be null");
        if (num > 1) {
            while (num % 2 == 0)
                num /= 2;
        }
        return num == 1;
    }
}
```

8. Извршување на сите досегашни тестови од сите 4 класи, користејќи TestSuite

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses( { PowerOfTwoCheckerTests.class,
                 PowerOfTwoCheckerTimeoutTests.class,
                 PowerOfTwoCheckerParameterizedTest.class,
                 PowerOfTwoCheckerExceptionTests.class} )
public class JUnitTestSuite {
}
```


<default package> 136 ms
JUnit Jupiter 131 ms
PowerOfTwoCheckerExceptionTests 31 ms
isPowerOfTwoShouldNotThrowAnExceptionTest()
isPowerOfTwoShouldThrowAnExceptionTest() 31 ms
isPowerOfTwoShouldThrowAnExceptionTestWithMessage()
PowerOfTwoCheckerParameterizedTest 67 ms
isPowerOfTwoParameterizedTest(int, boolean) 67 ms
[1] 32, true 57 ms
[2] 55, false 1 ms
[3] 200, false 2 ms
[4] -512, false 4 ms
[5] 0, false 2 ms
[6] 1, true 1 ms
PowerOfTwoCheckerTests 23 ms
testIsPowerOfTwo1() 4 ms
testIsPowerOfTwo1_1() 1 ms
testIsPowerOfTwo2() 2 ms
testIsPowerOfTwo2_2() 3 ms
testIsPowerOfTwo3() 5 ms
testIsPowerOfTwo3_2() 1 ms
testIsPowerOfTwo4() 2 ms
testIsPowerOfTwo4_2() 2 ms
testIsPowerOfTwo5() 1 ms
testIsPowerOfTwo5_2() 1 ms
testIsPowerOfTwo6()
testIsPowerOfTwo6_2() 1 ms
PowerOfTwoCheckerTimeoutTests 10 ms
testIsPowerOfTwoTimeout1() 8 ms
testIsPowerOfTwoTimeout2() 2 ms
JUnit Vintage 5 ms
JUnitTestSuite 5 ms
2: Favorites
7: Structure
Terminal
Build
4: Run
6: TODO